From the Internet AS level topology to other complex networks

András Gulyás

November 21, 2017

Contents

1 Requirements

You can complete this notebook in your browser without installing anything. However if you like to experience more offline then the code snipplets can be run in an R console. For this the install of R and the igraph R package is required.

2 Import the igraph library (http://igraph.org/r)

```
library(igraph)
```

```
Attaching package: 'igraph'
```

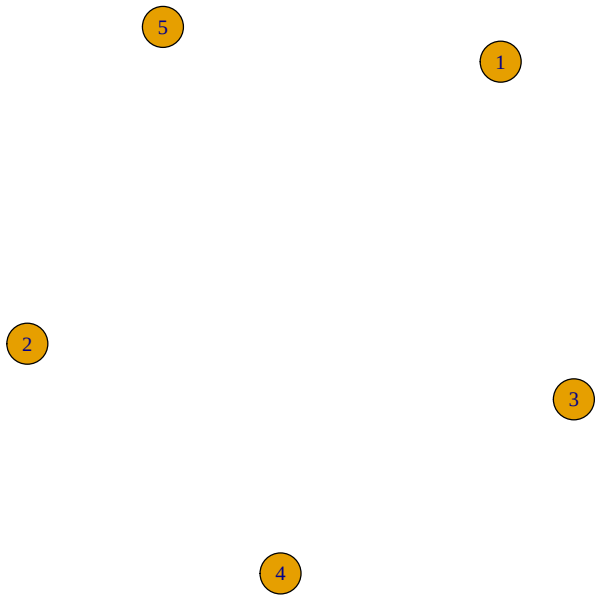
```
The following objects are masked from 'package:stats':

    decompose, spectrum
```

```
The following object is masked from 'package:base':

    union
```

3 Playing around with a sample graph

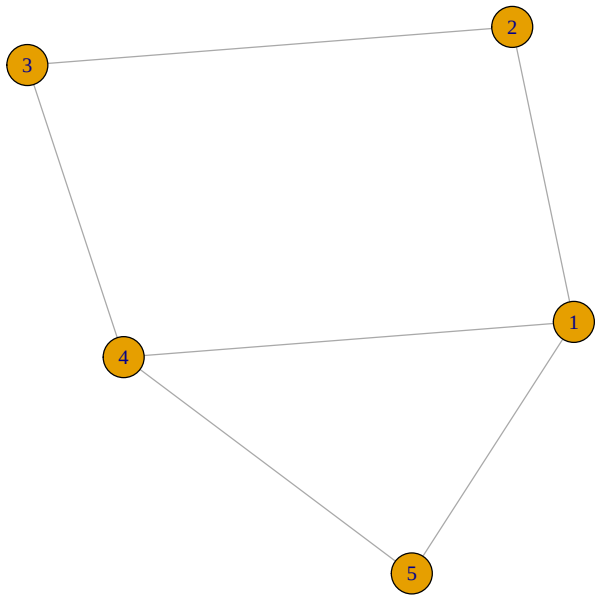
Create and plot an empty network with 5 vertices

```
g.sample<- make_empty_graph(5, directed=FALSE)
plot(g.sample)
```



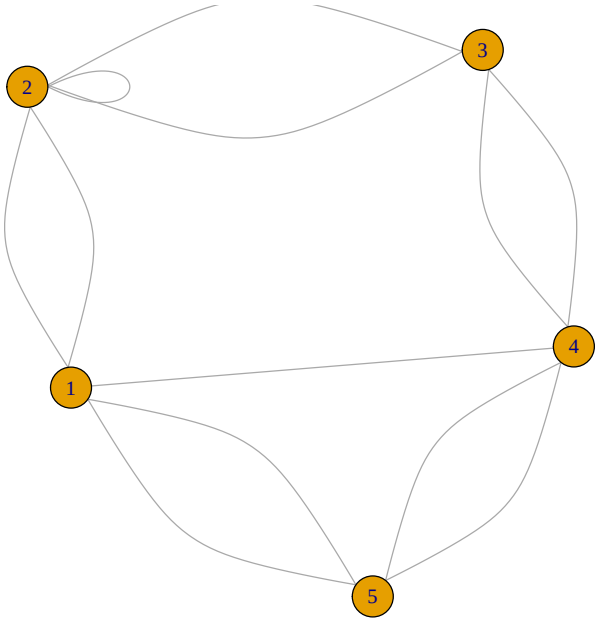
Add some edges

```
g.sample <- add_edges(g.sample,c(1,2,2,3,3,4,4,5,5,1,1,4))
g.sample_const <- g.sample # This is just for saving g.sample for future use
plot(g.sample)
```



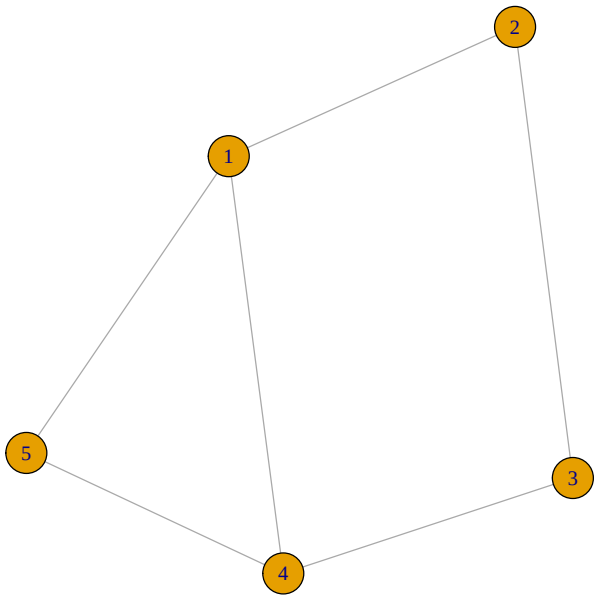
Add some more edges

```
g.sample <- add_edges(g.sample,c(1,2,2,3,3,4,4,5,5,1,2,2))
plot(g.sample)
```



Remove multiple and loop edges by simplify

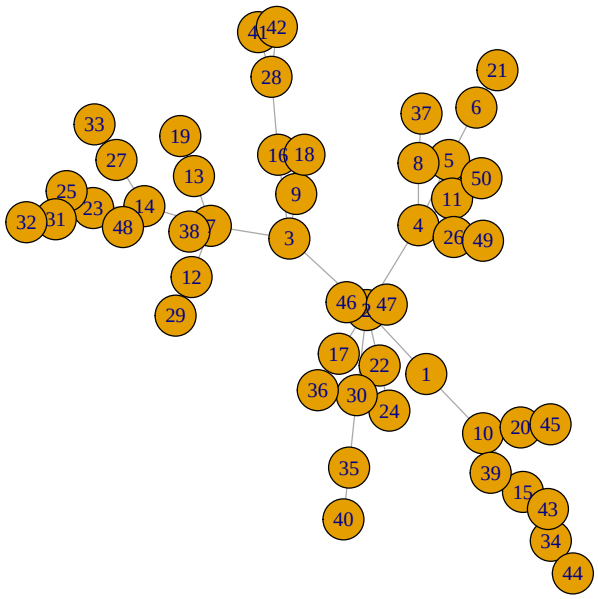
```
g.sample <- simplify(g.sample)
plot(g.sample)
```



3.1 Generate growing random tree

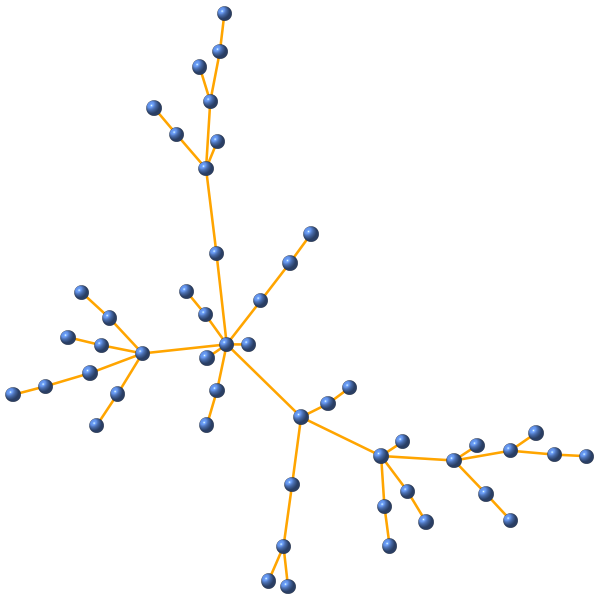
Random Tree Growth

```
N <- 50
random.growing.tree <- graph.empty(0,directed=FALSE)
random.growing.tree <- random.growing.tree + vertices(1)
 for (i in 2:N) {
   mate <- sample(V(random.growing.tree),1)
   random.growing.tree <- random.growing.tree + vertices(i)
   random.growing.tree <- random.growing.tree + edges(mate,i)
 }
plot(random.growing.tree)
```



A fancy plot

```
plot(random.growing.tree, vertex.size=5, vertex.label="",vertex.shape="sphere",
vertex.color="cornflowerblue",edge.color="orange",edge.width=2)
```



4 Getting info about the AS topology: goto http://routeviews.org

Open oix-damp-snapshot-2005-05-01-0000.dat to see BGP route dumps generated by running **show ip BGP** on the routers. Search for the path attribute in the ouput of the **show ip BGP** command.

```
## In emacs org
system("head -n200 oix-damp-snapshot-2005-05-01-0000.dat")
## Jupyter
## readChar("oix-damp-snapshot-2005-05-01-0000.dat",nchars=15000)

route-views.routeviews.org
spawn telnet route-views.routeviews.org
```

```
Trying 128.223.60.103...

Connected to route-views.routeviews.org.

Escape character is '^]'.


 ************************************************************************

                    Oregon Exchange BGP Route Viewer
          route-views.oregon-ix.net / route-views.routeviews.org

 route views data is archived on http://archive.routeviews.org

 This hardware is part of a grant from Cisco Systems.

 Please contact help@routeviews.org if you have questions or
 comments about this service, its use, or if you might be able to
 contribute your view.

 This router has views of the full routing tables from several ASes.
 The list of ASes is documented under "Current Participants" on
 http://www.routeviews.org/.

                          **************

 route-views.routeviews.org is now using AAA for logins.  Login with
 username "rviews".  See http://routeviews.org/aaa.html

 ************************************************************************


User Access Verification

Username: rvcoltr
Password:

route-views.oregon-ix.net>
route-views.oregon-ix.net>term length 0
route-views.oregon-ix.net>show ip bgp summary
```

```
BGP router identifier 198.32.162.100, local AS number 6447
BGP table version is 38856886, main routing table version 38856886
177841 network entries using 17961941 bytes of memory
7780861 path entries using 373481328 bytes of memory
1386984 BGP path attribute entries using 77708736 bytes of memory
1085000 BGP AS-PATH entries using 29280690 bytes of memory
14610 BGP community entries using 761440 bytes of memory
1 BGP extended community entries using 24 bytes of memory
0 BGP route-map cache entries using 0 bytes of memory
0 BGP filter-list cache entries using 0 bytes of memory
BGP using 499194159 total bytes of memory
Dampening enabled. 5212 history paths, 11648 dampened paths
BGP activity 350618/166826 prefixes, 47791725/39965835 paths, scan interval 60 secs

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
4.68.0.243      4  3356 2655592   76543 38856847    0    0 1w1d       156843
12.0.1.63       4  7018 3184031   45932 38856847    0    0 1w1d       157349
62.164.11.10    4  8782   76491   76518 38856847    0    0 4d00h           0
64.125.0.137    4  6461 3049226   86336 38856847    0    0 21:00:36   157929
64.200.95.239   4  7911 2360529   76550 38856847    0    0 1w1d       157662
64.200.151.12   4  7911 2457488   76543 38856847    0    0 1w1d       157662
65.106.7.139    4  2828 1852877   74628 38856847    0    0 1w1d       157383
66.185.128.48   4  1668 2012271   76551 38856847    0    0 1w1d       158010
83.88.48.13     4  3292 2221870   46035 38856847    0    0 1w1d       159029
129.250.0.11    4  2914 5200326  152192 38856847    0    0 2d21h      157775
129.250.0.85    4  2914 4928186  153056 38856847    0    0 2d21h      157775
134.55.200.1    4   293 5327378  153093 38856847    1    0 4d01h      159112
134.222.85.45   4   286 5016164   76543 38856847    2    0 2d20h      158155
141.142.12.1    4  1224  153082  153085 38856847    0    0 5d10h           4
144.228.241.81  4  1239 2881272   45926 38856847    0    0 1w1d       157169
164.128.32.11   4  3303 1022309   45932 38856847    0    0 1w1d        71714
167.142.3.6     4  5056 3089397   76562 38856847    0    0 5d10h      157652
193.0.0.56      4  3333 1716214   76723 38856847    0    0 2w5d       160834
193.251.245.6   4  5511 3316677   45931 38856847    0    0 1w1d       157673
194.85.4.55     4  3277 3380863   45931 38856847    0    0 2d15h      160377
195.66.224.82   4  4513 3936653   76536 38856847    0    0 1w1d       159480
195.66.232.239  4  5459 1305866   45931 38856847    0    0 1w1d        78270
195.219.96.239  4  6453 2793489   45931 38856847    0    0 1w1d       157664
196.7.106.245   4  2905 3104797   76541 38856847    0    0 1w1d       165145
198.32.8.196    4 11537  371728     466 38856847    0    0 1w1d         9413
```

```
198.32.162.1    4  3582  153375  153432 38856847    0    0 5d01h           1
202.232.0.2     4  2497 3778622   45931 38856847    0    0 1w1d       158204
202.249.2.86    4  7500 2729977   64154 38856847    0    0 1w1d       163760
203.62.252.26   4  1221 2222048   45931 38856847    0    0 1w1d       158712
203.181.248.233 4  7660 2013965     466 38856847    1    0 1w1d       160964
204.42.253.253  4   267  422786   45875 38856847    0    0 1w1d        10649
205.189.32.44   4  6509  371722   22969 38856847    0    0 1w1d        10012
206.24.210.26   4  3561 2535560     470 38856847    0    0 1w1d       157032
206.186.255.223 4  2493 1591969   45932 38856847    0    0 1w1d       158021
206.220.240.222 4 10764  477089   47878 38856847    0    0 5d10h        9571
207.45.223.244  4  6453 2848419   45931 38856847    0    0 1w1d       157513
207.46.32.32    4  8075 2137580   76542 38856847    0    0 1w1d       158278
207.172.6.162   4  6079 2071517   76562 38856847    0    0 2d10h      146191
207.172.6.227   4  6079 3547134   76570 38856847    0    0 2d10h      146193
207.246.129.6   4 11608 1411024   28500 38856847    0    0 1w1d       158069
207.246.129.14  4 11608 3189551   76537 38856847    0    0 1w1d       158327
208.51.134.254  4  3549 6489015     470 38856847    0    0 1w1d       157645
208.186.154.35  4  5650 2415510  153099 38856847    0    0 4d01h      157887
208.186.154.36  4  5650 2534274  153102 38856847    0    0 4d18h      157887
209.10.12.28    4  4513 4335176   76545 38856847    0    0 1w1d       159362
209.10.12.125   4  4513 4822012   76546 38856847    0    0 1w1d       159360
209.10.12.156   4  4513 4311146   76549 38856847    0    0 3d09h      159368
209.123.12.51   4  8001 8163163  153442 38856847    0    0 2w5d       158552
209.161.175.4   4 14608 3123916   55081 38856847    0    0 1w6d       160195
213.140.32.146  4 12956 6305837   76548 38856847    0    0 1w1d       157900
213.200.87.254  4  3257 1472545     470 38856847    0    0 1w1d       157686
213.248.83.252  4  1299 3723901   76558 38856847    0    0 2d21h      157197
216.18.63.137   4  6539 2334641   76552 38856847    0    0 1w1d       158184
216.140.2.59    4  6395 4552884  153108 38856847    0    0 4d01h      157853
216.140.8.59    4  6395 4331066  153115 38856847    0    0 5d10h      157843
216.218.252.145 4  6939 4298436   76549 38856847    0    0 1w1d       158827
217.75.96.60    4 16150 2921757   76561 38856847    0    0 1w1d       158324
route-views.oregon-ix.net>show ip bgp dampening dampened-paths
BGP table version is 38856887, local router ID is 198.32.162.100
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          From             Reuse    Path
*d 195.153.124.0    194.85.4.55      00:00:10 3277 13062 25462 3257 1239 2914 174 31742
```

```
*d 83.137.224.0/21  194.85.4.55      00:00:10 3277 13062 25462 3257 1239 2914 174 31742
*d 203.14.208.0     216.18.63.137    00:00:10 6539 3561 7473 7474 10223 i
*d 203.149.67.0     216.18.63.137    00:00:10 6539 3561 7473 7474 10223 17766 i
*d 203.149.66.0     216.18.63.137    00:00:10 6539 3561 7473 7474 10223 17766 i
*d 203.149.65.0     216.18.63.137    00:00:10 6539 3561 7473 7474 10223 17766 i
*d 61.68.240.0/20   216.18.63.137    00:00:10 6539 2914 1239 4648 2764 i
*d 61.68.192.0/20   216.18.63.137    00:00:10 6539 2914 1239 4648 2764 i
*d 210.10.96.0/22   216.18.63.137    00:00:10 6539 3561 7473 7474 9556 17455 i
*d 203.161.156.0    216.18.63.137    00:00:10 6539 3561 7473 7474 9556 17455 i
*d 203.23.233.0     216.18.63.137    00:00:10 6539 3561 7473 7474 9556 17455 i
*d 202.14.187.0     216.18.63.137    00:00:10 6539 3561 7473 7474 9556 17455 i
*d 168.217.0.0      216.18.63.137    00:00:10 6539 3561 7473 7474 18292 i
*d 203.119.14.0/23  216.18.63.137    00:00:10 6539 2914 1239 4648 2764 18210 i
*d 203.57.3.0       216.18.63.137    00:00:10 6539 2914 1239 4648 2764 18210 i
*d 203.18.56.0      216.18.63.137    00:00:10 6539 2914 1239 4648 2764 18210 i
*d 202.65.12.0/22   216.18.63.137    00:00:10 6539 2914 1239 4648 2764 18210 i
*d 203.222.104.0    216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 ?
*d 218.185.0.0/17   216.18.63.137    00:00:10 6539 3561 7473 7474 10223 i
*d 203.94.128.0/18  216.18.63.137    00:00:10 6539 3561 7473 7474 10223 i
*d 203.27.50.0      216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7492 i
*d 203.22.247.0     216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7492 i
*d 203.11.129.0     209.123.12.51    00:00:10 8001 19151 7473 7474 10223 11001 i
*d 12.151.140.0/24  213.140.32.146   00:00:10 12956 3356 7018 2386 i
*d 192.160.244.0    207.172.6.227    00:00:10 6079 7911 22742 i
*d 192.160.243.0    207.172.6.227    00:00:10 6079 7911 22742 i
*d 214.13.175.0     209.10.12.125    00:00:10 4513 701 209 721 568 1733 1733 1733 1733
*d 214.13.174.0     209.10.12.125    00:00:10 4513 701 209 721 568 1733 1733 1733 1733
*d 214.13.173.0     209.10.12.125    00:00:10 4513 701 209 721 568 1733 1733 1733 1733
*d 12.39.152.0/24   64.125.0.137     00:00:10 6461 1239 26973 26973 26973 26973 26973 2
*d 12.39.154.0/23   64.125.0.137     00:00:10 6461 1239 26973 26973 26973 26973 26973 2
*d 213.243.63.0     216.140.2.59     00:00:10 6395 1299 9121 15924 16187 i
*d 195.244.32.0/19  134.55.200.1     00:00:10 293 1299 9121 i
*d 81.213.0.0/17    134.55.200.1     00:00:10 293 1299 9121 9121 i
*d 212.15.9.0       64.125.0.137     00:00:10 6461 1273 9121 15924 12873 i
*d 62.201.192.0/24  216.140.8.59     00:00:10 6395 3356 22351 6822 5422 25574 i
*d 213.153.239.0    64.125.0.137     00:00:10 6461 8447 8513 8513 8513 8513 8705 31090
 d 214.13.160.0/22  209.123.12.51    00:00:10 8001 701 209 721 568 1733 5803 5803 5803
*d 195.178.10.0     209.10.12.125    00:00:20 4513 701 702 8413 i
*d 210.156.112.0/20 216.18.63.137    00:00:20 6539 6461 2519 18285 23630 ?
*d 210.134.192.0/20 216.18.63.137    00:00:20 6539 6461 2519 18285 23630 ?
```

```
*d 203.34.175.0     216.18.63.137    00:00:10 6539 2914 701 703 7579 7579 ?
*d 210.9.235.0      216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.222.124.0/22 216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.222.123.0    216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.222.120.0/22 216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.222.116.0/22 216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.222.104.0/21 216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.222.96.0/21  216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.222.87.0     216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.222.86.0     216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.222.84.0/23  216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.222.83.0     216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.222.80.0/22  216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.222.78.0     216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.222.76.0/23  216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.222.72.0/22  216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.222.68.0/22  216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.222.64.0/22  216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.122.188.0/23 216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.122.184.0/22 216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.122.160.0/20 216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.122.152.0/21 216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.122.128.0/20 216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.63.200.0     216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.56.108.0     216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.55.43.0      216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.34.173.0     216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.34.170.0     216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.34.169.0     216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.34.168.0     216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.34.155.0     216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.31.254.0     216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.17.212.0     216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 203.12.22.0      216.18.63.137    00:00:10 6539 2914 1239 4648 2764 7579 7579 7579 1
*d 61.68.96.0/19    216.18.63.137    00:00:10 6539 2914 1239 4648 2764 i
*d 61.68.80.0/21    216.18.63.137    00:00:10 6539 2914 1239 4648 2764 i
*d 195.225.132.0/23 134.222.85.45    00:00:20 286 8928 25307 15498 i
*d 203.124.243.0    216.140.8.59     00:00:10 6395 3356 4755 9238 i
*d 203.124.197.0    216.140.8.59     00:00:10 6395 3356 4755 9238 i
*d 212.174.128.0/17 134.55.200.1     00:00:10 293 1299 9121 i
```

```
*d 195.244.48.0/20  216.140.8.59     00:00:10 6395 1299 9121 i
```

From these dumps we can extract the AS path argument for each prefix with simple text processing (not shown here). Lets suppose we have some paths and construct a network using these paths.

Begin with one path:

```
path1 <- c(1,2,3,4)
```

Now we have to convert this to an edgelist. Let's create two vectors with the beginning and the end of path1:

```
path1 <- c(1,2,3,4)
begin <- path1[-length(path1)]
end <- path1[-1]
cat("begin", begin, "end", end, "\n")

begin 1 2 3 end 2 3 4
```

and "ZIP" them as thus we get c(1,2,2,3,3,4), the edges on path1.

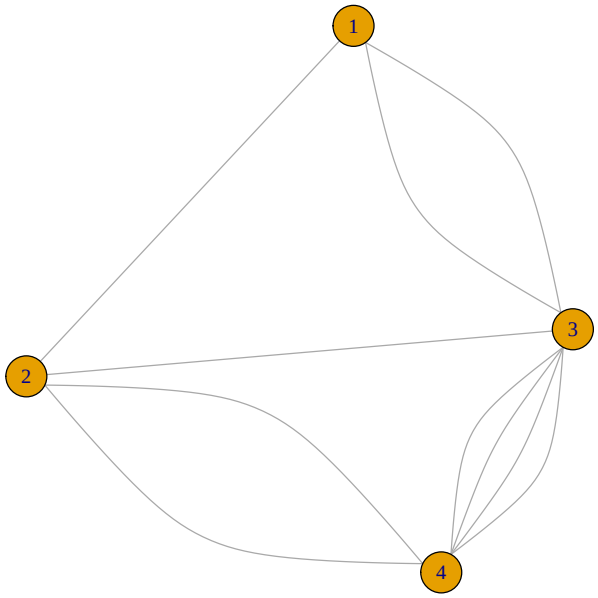
```
edges <- as.vector(rbind(begin,end))
cat(edges, "\n")

1 2 2 3 3 4
```

Now do this for all of our paths and construct the graph using the edgelists.

```
path1 <- c(1,2,3,4)
edges <- as.vector(rbind(path1[-length(path1)], path1[-1] ))
path2 <- c(2,4,3)
## Concatenate the edges of path2 to the edges of path1
edges <- c(edges, as.vector(rbind(path2[-length(path2)], path2[-1] )))
path3 <- c(1,3,4)
edges <- c(edges, as.vector(rbind(path3[-length(path3)], path3[-1] )))
path4 <- c(1,3,4,2)
edges <- c(edges, as.vector(rbind(path4[-length(path4)], path4[-1] )))

cat(edges, "\n")

1 2 2 3 3 4 2 4 4 3 1 3 3 4 1 3 3 4 4 2
```

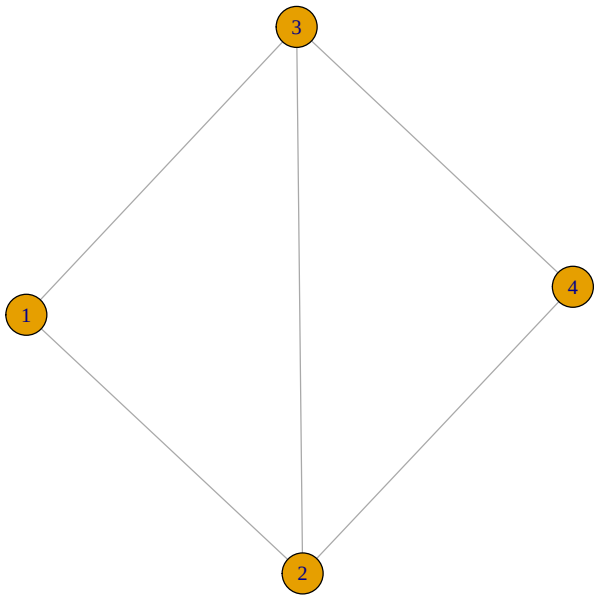
Now create the network using the edges.

```
## Create empty network with max(edges) vertices. max(edges)
## gives the highest vertex id in this case

g.path <- make_empty_graph(max(edges), directed=FALSE)
g.path <- add_edges(g.path,edges)
plot(g.path)
```



Remove multiple edges:

```
g.path <- simplify(g.path)
plot(g.path)
```



From a similar processing of many BGP dumps we recover an approximate picture of the Internet AS topology. In this class we use a a smaller historic network from 2010 ((asgraph.gml)).

5 Load the AS graph

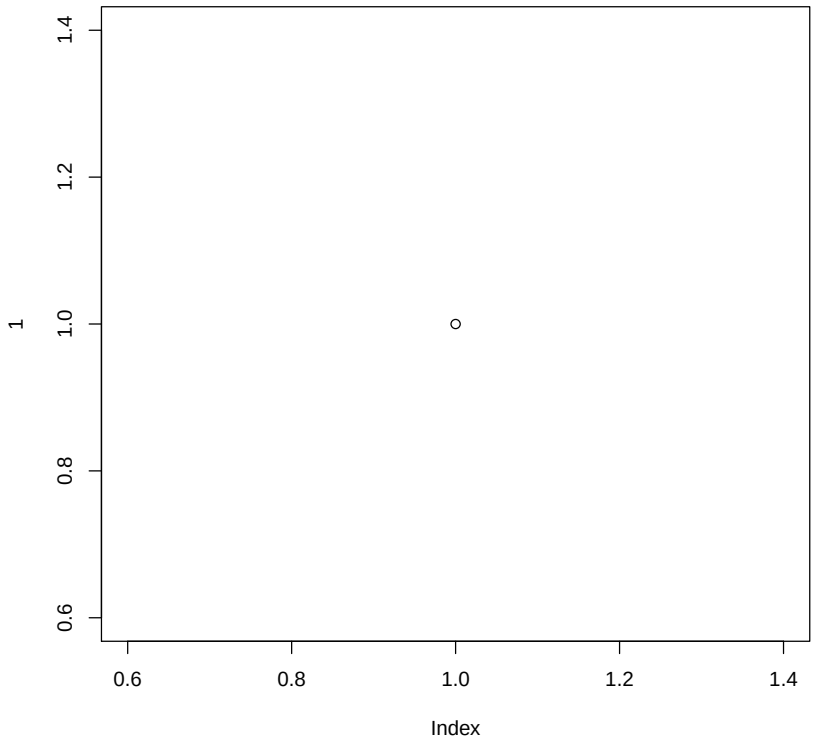
```
asgraph<-read.graph("asgraph.gml", format="gml")
asgraph
```

```
 attr: id (v/n), name (v/c), radius (v/n), angle (v/n), weight (e/n)
+ edges (vertex names):
 [1] 10010--18139 10010--23615 10010--23785 10010--23795 10010--24268
 [6] 10010--7690  10010--9615  10012--23720 10015--10002 10015--17945
[11] 10015--18264 10015--9351  10015--9997  10021--18265 10021--23820
[16] 10021--23940 10021--38462 10021--41935 10021--7503  10021--7678
[21] 10021--9824  10026--10024 10026--10030 10026--10032 10026--10083
```

```
[26] 10026--10084 10026--10126 10026--10135 10026--10143 10026--10145
[31] 10026--10157 10026--10171 10026--10236 10026--10310 10026--109
[36] 10026--11351 10026--1239  10026--12601 10026--1273  10026--1299
+ ... omitted several edges
```

This is only on friday evenings

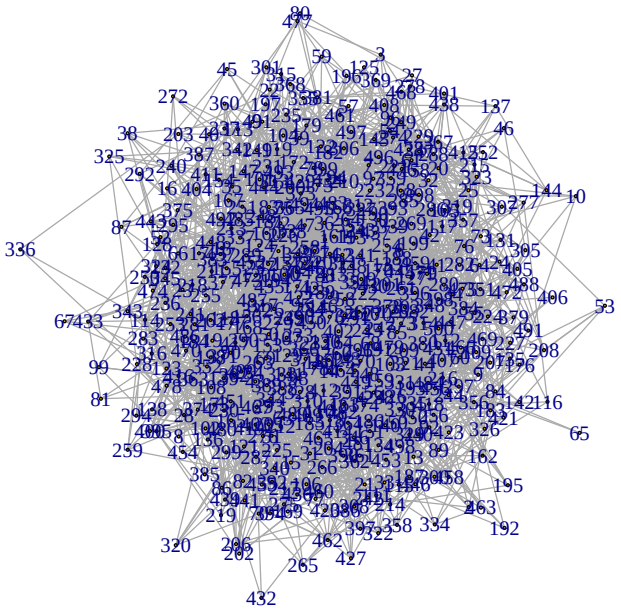
```
# plot(asgraph) ## Hit C-c C-c, we don't have time for this
plot(1)
```



Check what we can see by plotting large graphs. Random network without structure

```
N <- 500
k <- 10
p <- k/N
er <- erdos.renyi.game(N, p)
```

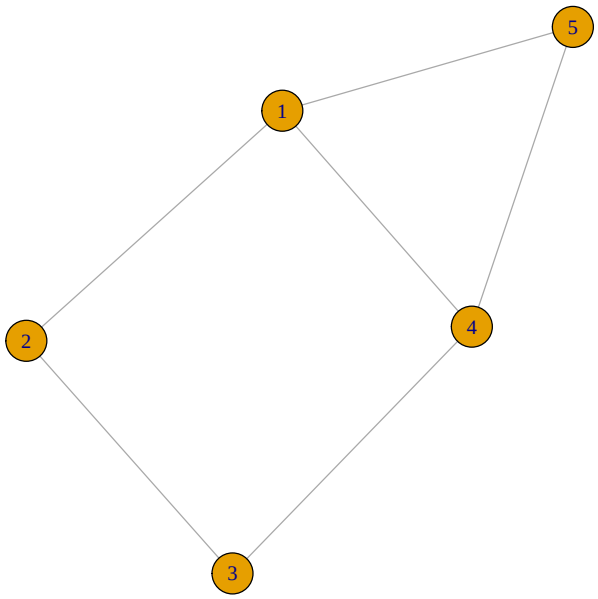
```
cl <- clusters(er)
gi <- induced.subgraph(er, which(cl$membership == which.max(cl$csize)))
plot(gi,vertex.size=1)
```



6 Is it a connected graph?

See the connected components of a network. Revert the sample graph first.

```
## Revert our sample graph
g.sample <- g.sample_const
plot(g.sample)
```

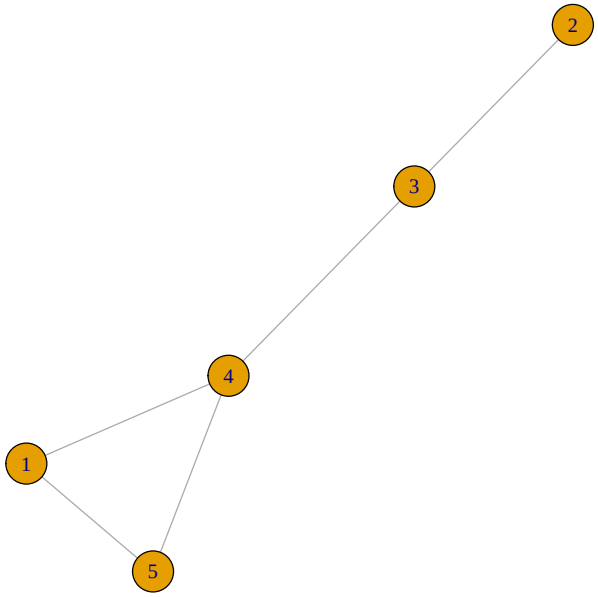


Compute connected components.

```
clusters(g.sample)

$membership
[1] 1 1 1 1 1

$csize
[1] 5

$no
[1] 1
```

Delete and edge.

```
g.sample<- delete_edges(g.sample,"1|2")
plot(g.sample)
```

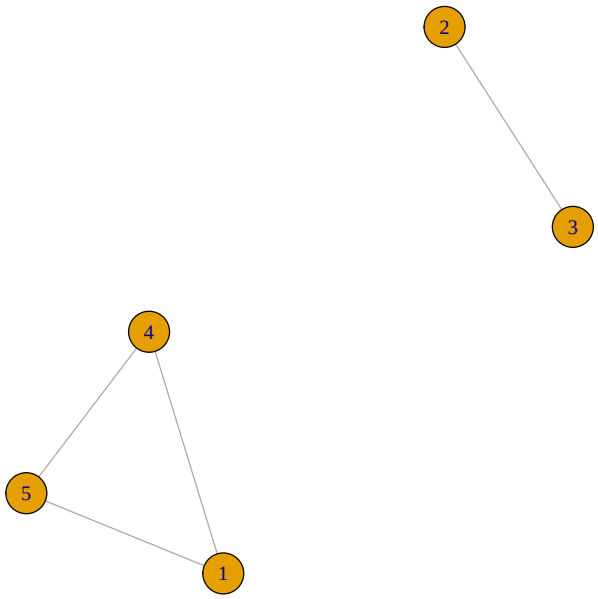


Recompute connected components.

```
clusters(g.sample)

$membership
[1] 1 1 1 1 1

$csize
[1] 5

$no
[1] 1
```

Remove one more edge.

```
g.sample<- delete_edges(g.sample,"3|4")
plot(g.sample)
```



Recompute connected components.

```
clusters(g.sample)
```

```
$membership
[1] 1 2 2 1 1

$csize
[1] 3 2

$no
[1] 2
```

All rigth let's compute this for the clusters for the AS graph

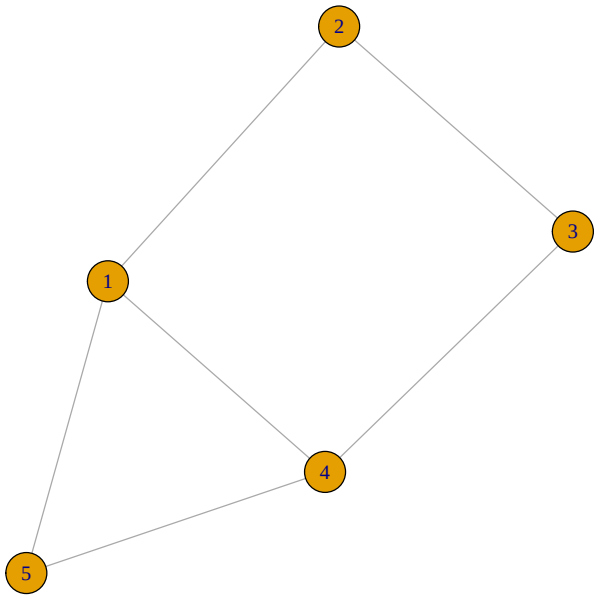
```
cl<-clusters(asgraph)
cl$csize
cl$no
```

```
[1] 23748
[1] 1
```

7 Is it sparse or dense?

Compute the number of vertices, edges and average degree for our sample graph.

```
## Revert our sample graph
g.sample <- g.sample_const
plot(g.sample)
```



Compute the number of edges, nodes, average degree.

```
ecount(g.sample)
vcount(g.sample)
(ecount(g.sample)*2)/vcount(g.sample)
```

```
[1] 6
[1] 5
[1] 2.4
```

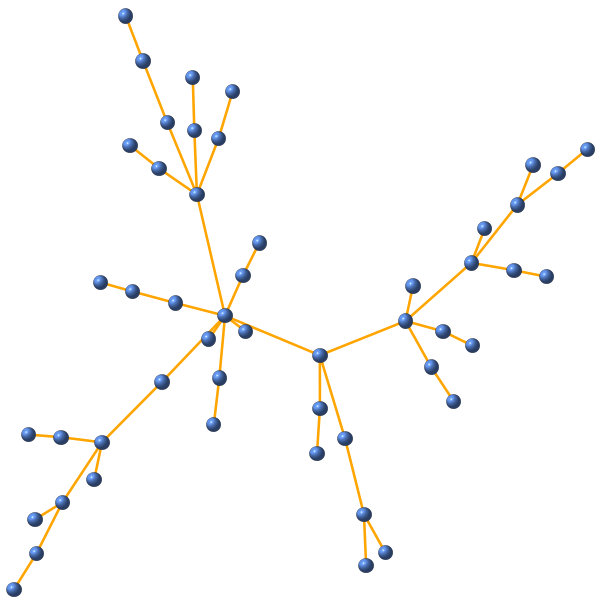
And for the internet

```
ecount(asgraph)
vcount(asgraph)
(ecount(asgraph)*2)/vcount(asgraph)
```

```
[1] 58414
[1] 23748
[1] 4.919488
```

Compare to a tree

```
plot(random.growing.tree, vertex.size=5, vertex.label="",vertex.shape="sphere",
vertex.color="cornflowerblue",edge.color="orange", edge.width=2)
```



Comapre average degree.

```
(ecount(random.growing.tree)*2)/vcount(random.growing.tree)
```

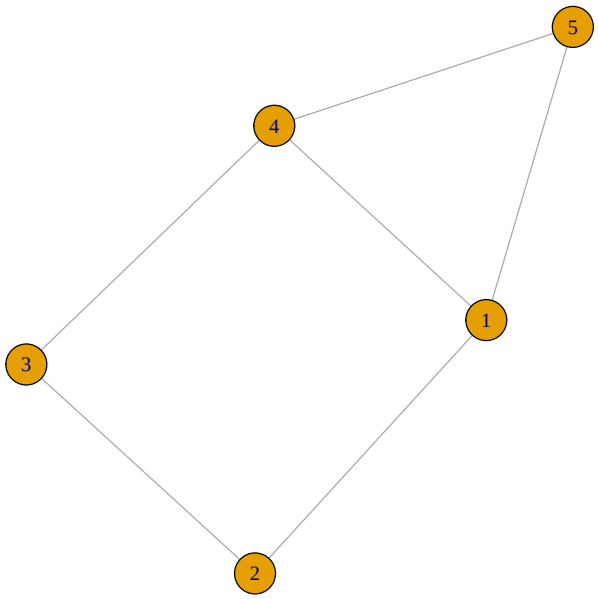
```
[1] 1.96
```

8 On the Internet short paths are vital. What is the diameter and average distance in the Internet?

8.1 What is diameter and average distance?

The diameter is the length of the longest shortest path in the network. Let's compute this manually to feel it.

```
## Revert our sample graph
g.sample <- g.sample_const
plot(g.sample)
```



Compute the length of the shortest path between two vertices

```
distances(g.sample, 1, 2, mode = c("all"), algorithm = "dijkstra")
```

```
     [,1]
[1,]    1
```

Compute the length of the shortest path between one and many other vertices

```
distances(g.sample, 1, c(1,2,3,4,5), mode = c("all"), algorithm = "dijkstra")
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    2    1    1
```

Compute the length of the shortest paths between all pairs, then compute diameter and average path length.

```
dists <- distances(g.sample, c(1,2,3,4,5), c(1,2,3,4,5), mode = c("all"), algorithm = "
max(dists)
mean(dists)
```

```
[1] 2
[1] 1.12
```

8.2 Compute these in a more convenient way

Compute the diameter of a graph.

```
diameter(g.sample)
average.path.length(g.sample, directed=FALSE)
```

```
[1] 2
[1] 1.4
```

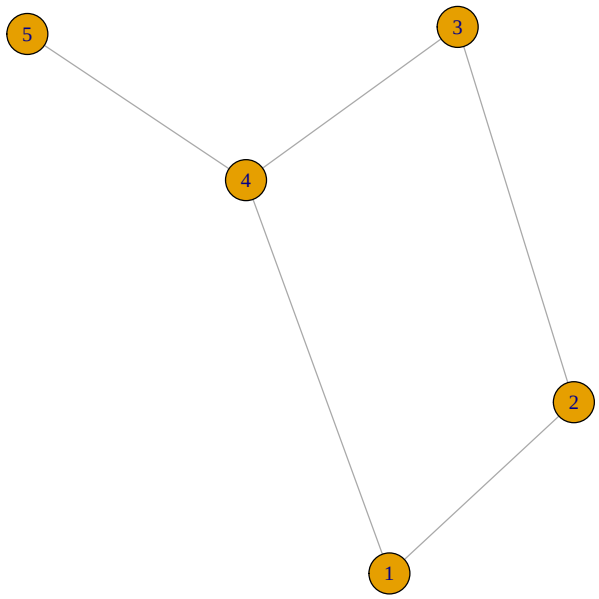
Compute mean by excluding zeros

```
## We should exclude trivial zeros
mean(dists[dists!=0])
```

```
[1] 1.4
```

Alter the network a bit and see the changes

```
g.sample<- delete_edges(g.sample,"5|1")
plot(g.sample)
```



Recompute diameter and average path length

```
diameter(g.sample)
average.path.length(g.sample, directed=FALSE)
```

```
[1] 3
[1] 1.6
```

8.3 Diameter of the Internet AS level graph

This will take a while

```
diam <- diameter(asgraph,weights=NA)
diam
avgdist <- average.path.length(asgraph, directed=FALSE)
avgdist
```

```
[1] 9
[1] 3.521532
```

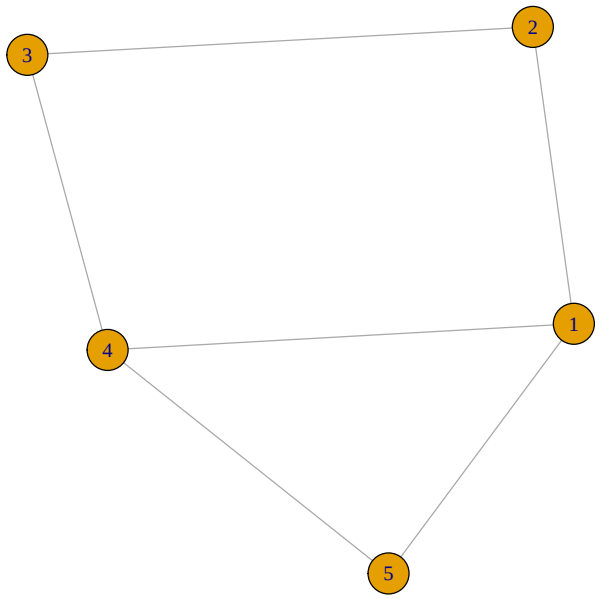
Stop for a moment and think about this for a while. A large sparse network with that small diemeter? How can it be?

9 Clustering: the friend of my friend is my friend too?

9.1 Manually compute clustering

See https://en.wikipedia.org/wiki/Network_science#Clustering_coefficient for the defintion of the clustering coefficient.

```
## Revert our sample graph
g.sample <- g.sample_const
plot(g.sample)
```



Compute the connected triplets centered on 1

```
deg <- degree(g.sample,1)
number.of.triplets.centered.on.1 <- deg*(deg-1)/2
```

Check the edges between i's neighbors and compute its local clustering.

```
count <- 0
neighbors <- neighbors(g.sample,1)
possible.friendsips <- combn(neighbors,2)
for (pf in 1:ncol(possible.friendsips)){
    friend1 <- possible.friendsips[,pf][1]
    friend2 <- possible.friendsips[,pf][2]
    if(are_adjacent(g.sample, friend1, friend2)){
        count <- count + 1
    }
}
```

```
cat("Local clustering of 1 is",count/number.of.triplets.centered.on.1,"\n\n")
```

```
Local clustering of 1 is 0.3333333
```

Compute this for all vertices in the network

```
for (i in V(g.sample)){
    cat("Vertex", i, "\n")
    ## Compute the connected triplets centered on i
    deg <- degree(g.sample,i)
    number.of.triplets.centered.on.i <- deg*(deg-1)/2
    ## Get the edge between i's neighbors
    count <- 0
    neighbors <- neighbors(g.sample,i)
    print(neighbors)
    possible.friendsips <- combn(neighbors,2)
    print(possible.friendsips)
    for (pf in 1:ncol(possible.friendsips)){
        friend1 <- possible.friendsips[,pf][1]
        friend2 <- possible.friendsips[,pf][2]
        if(are_adjacent(g.sample, friend1, friend2)){
            count <- count + 1
        }
    }
    cat("Local clustering:", count/number.of.triplets.centered.on.i,"\n\n")
}
```

```
 Vertex 1
+ 3/5 vertices:
[1] 2 4 5
     [,1] [,2] [,3]
[1,]    2    2    4
[2,]    4    5    5
Local clustering: 0.3333333

Vertex 2
+ 2/5 vertices:
[1] 1 3
     [,1]
[1,]    1
```

```
[2,]    3
Local clustering: 0

Vertex 3
+ 2/5 vertices:
[1] 2 4
     [,1]
[1,]    2
[2,]    4
Local clustering: 0

Vertex 4
+ 3/5 vertices:
[1] 1 3 5
     [,1] [,2] [,3]
[1,]    1    1    3
[2,]    3    5    5
Local clustering: 0.3333333

Vertex 5
+ 2/5 vertices:
[1] 1 4
     [,1]
[1,]    1
[2,]    4
Local clustering: 1
```

9.2 Using built-in function

```
transitivity(g.sample,type="local")

[1] 0.3333333 0.0000000 0.0000000 0.3333333 1.0000000
```

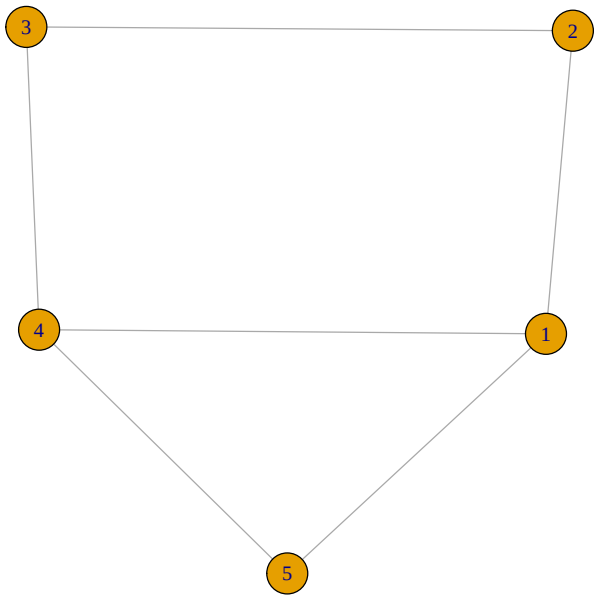
Let's compute clustering for the asgraph

```
clustering<-transitivity(asgraph,type="local")
clustering[1:100] # show only the first few results.

 [1] 0.120879121 1.000000000 0.000000000 0.000000000         NaN 0.000000000
 [7] 0.333333333 0.500000000 0.666666667 0.333333333 0.214285714 0.666666667
[13]         NaN 0.000000000 0.166666667 0.666666667 0.163636364 1.000000000
```

```
[19] 0.000000000 0.000000000 0.333333333 0.500000000 0.666666667 0.000000000
[25] 0.143790850 0.034727303 1.000000000         NaN 0.400000000 0.666666667
[31] 0.200000000 0.476190476 0.500000000 0.527777778 1.000000000 0.476190476
[37] 1.000000000 1.000000000 0.144501279 0.478947368 0.059891107 0.005607334
[43] 0.000000000 0.020465911 0.009156548 0.523809524 0.722222222 0.457607433
[49] 0.119898422 0.143434343 0.461538462 0.003037295 0.357142857 0.500000000
[55] 0.000000000 0.485714286 0.110661269 1.000000000 0.166666667         NaN
[61] 0.191666667 0.400000000 0.300000000         NaN 0.428571429 0.545454545
[67]         NaN 0.400000000 0.252016129 0.203463203 0.230769231 0.333333333
[73] 0.666666667 1.000000000 0.218181818 0.444444444 0.333333333 0.000000000
[79] 0.102564103 0.162162162 0.116666667 1.000000000 0.678571429         NaN
[85] 0.138888889 0.333333333 0.833333333 0.333333333 0.309090909 0.333333333
[91] 1.000000000         NaN 0.666666667 0.333333333 1.000000000 0.000000000
[97] 1.000000000 0.166666667 0.333333333         NaN
```

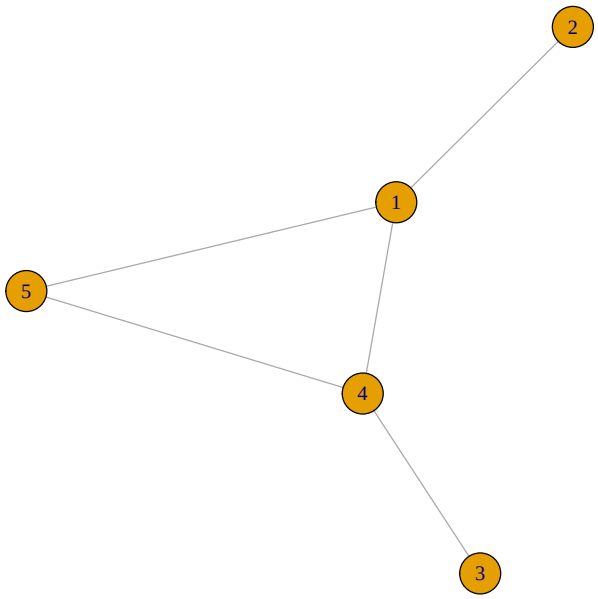
See what we get. What are these Nan's?

```
## Revert our sample graph
g.sample <- g.sample_const
plot(g.sample)
```



Remove and edge

```
g.sample<- delete_edges(g.sample,"2|3")
plot(g.sample)
```



Recompute clustering

```
transitivity(g.sample,type="local")
```

```
[1] 0.3333333       NaN       NaN 0.3333333 1.0000000
```

All right, get rid of Nan's and compute the clustring of the whole graph.

```
asgraph.clustering <- mean(clustering, na.rm=TRUE)
asgraph.clustering
```

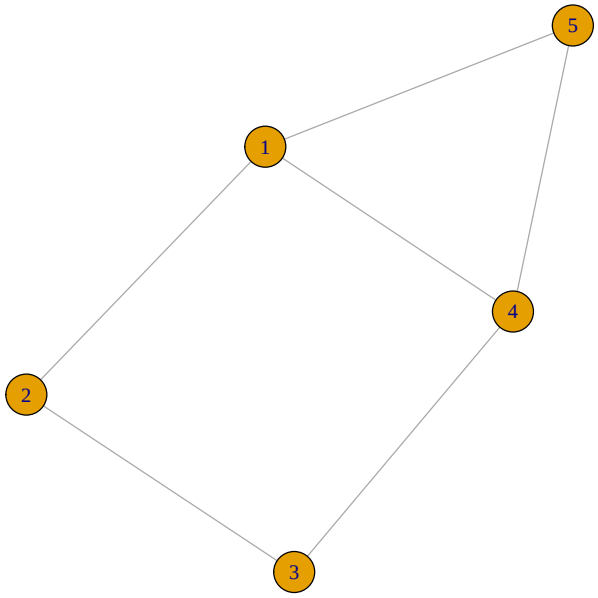
```
[1] 0.6055093
```

Think about this for a while. How can, a large, clustered network have small diameter.

10 Degree distribution of the nodes

10.1 Manually compute degree distribution

```
## Revert our sample graph
g.sample <- g.sample_const
plot(g.sample)
```



Let's compute the degrees of vertices.

```
x <- array()
y <- array()

for (v in V(g.sample)){
    print(degree(g.sample,v))
}
```

```
[1] 3
```

```
[1] 2
[1] 2
[1] 3
[1] 2
```

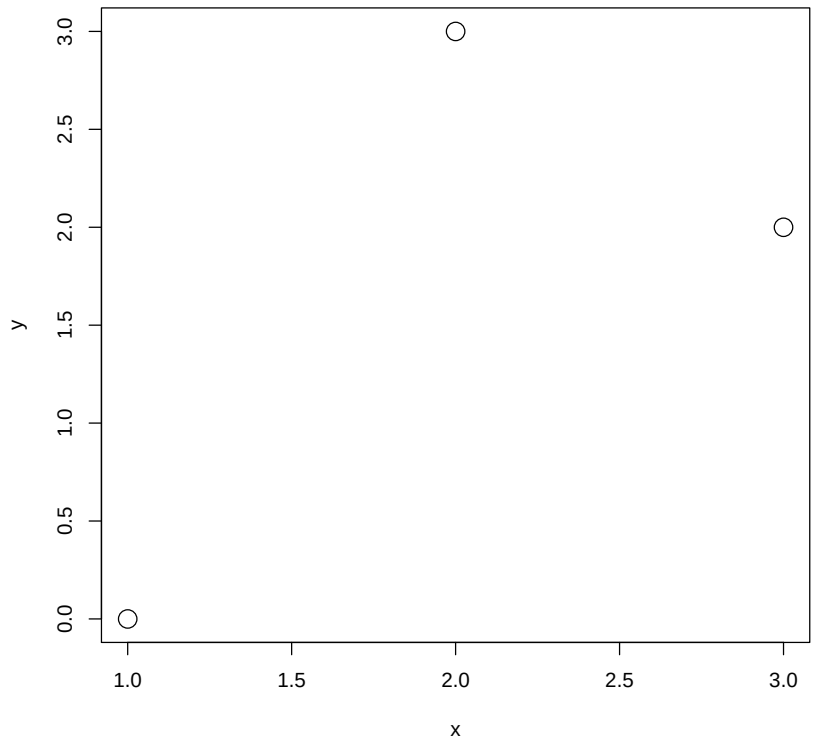
More simply.

```
degree(g.sample)
```

```
[1] 3 2 2 3 2
```

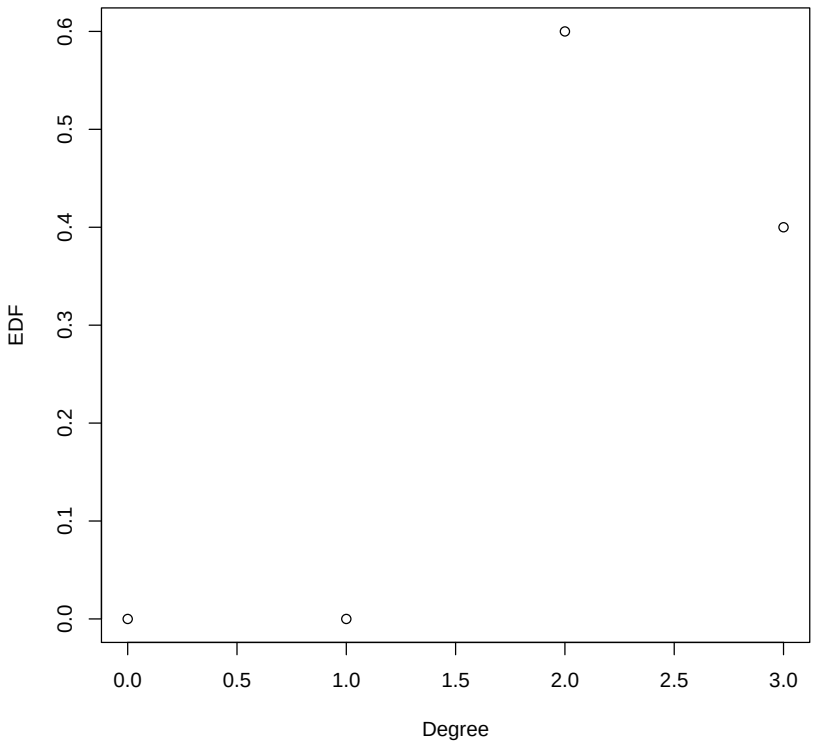
Now compute the degree distribution

```
x <- 1:max(degree(g.sample))
for (k in 1:max(degree(g.sample))){
    y[k] <- sum( degree(g.sample) == k )
}
y

plot(x,y,cex=2)
```



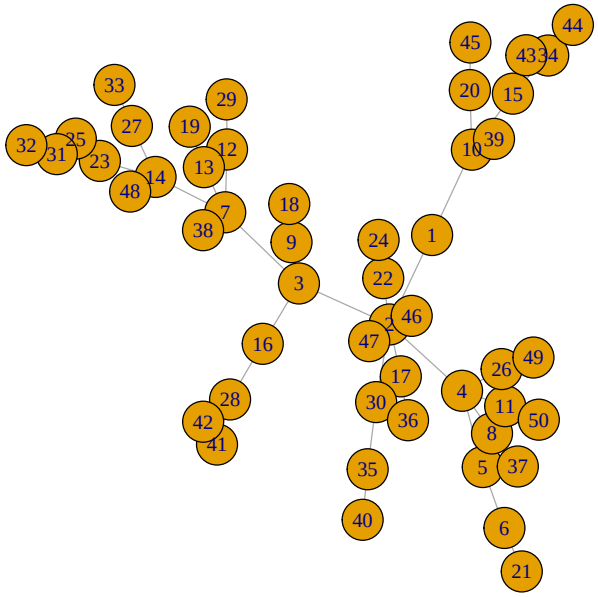
10.2 Using built in functions

```
dd<-degree.distribution(g.sample,mode="total")
max_degree<-max(degree(g.sample,mode="total"))
plot(0:max_degree,dd,xlab="Degree",ylab="EDF")
```



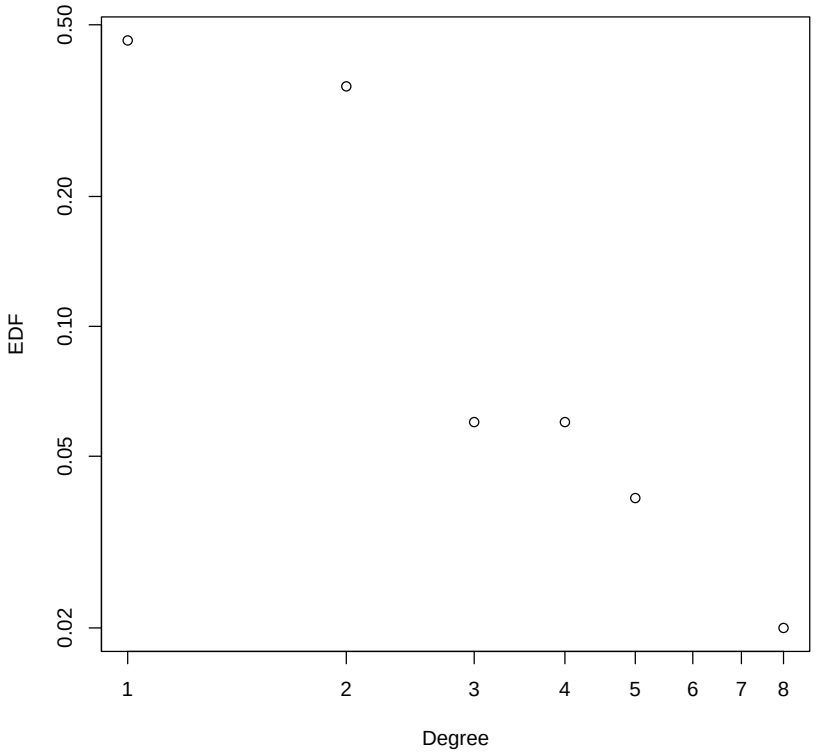
Plot our previously defined growing tree

```
plot(random.growing.tree)
```



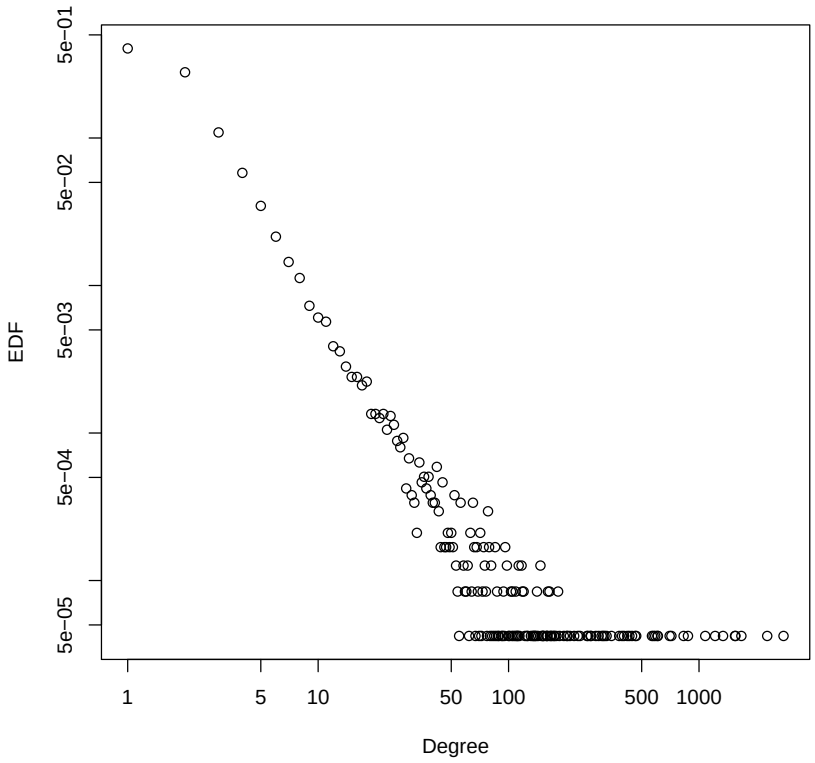
Plot its degree distribution

```
dd<-degree.distribution(random.growing.tree,mode="total")
max_degree<-max(degree(random.growing.tree,mode="total"))
plot(0:max_degree,dd,xlab="Degree",ylab="EDF",log="xy")
```



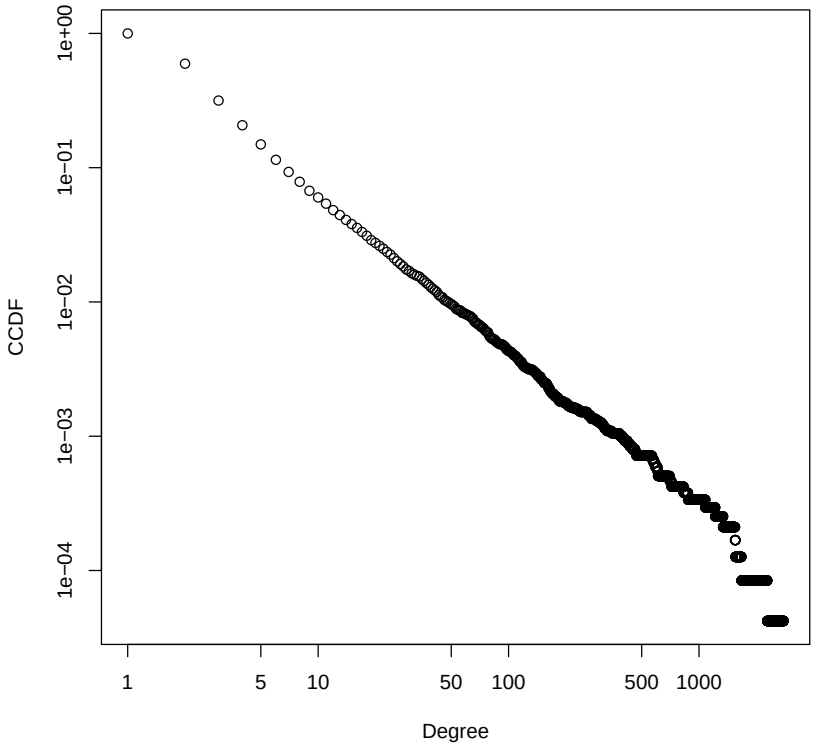
Degree distribution of the AS graph

```
dd<-degree.distribution(asgraph,mode="total")
max_degree<-max(degree(asgraph,mode="total"))
plot(0:max_degree,dd,xlab="Degree",ylab="EDF",log="xy")
```



EDF means empirical density function

```
dd_as<-degree.distribution(asgraph,mode="total",cumulative=TRUE)
max_degree_as<-max(degree(asgraph,mode="total"))
plot(0:max_degree_as,dd_as,xlab="Degree",ylab="CCDF",log="xy")
```

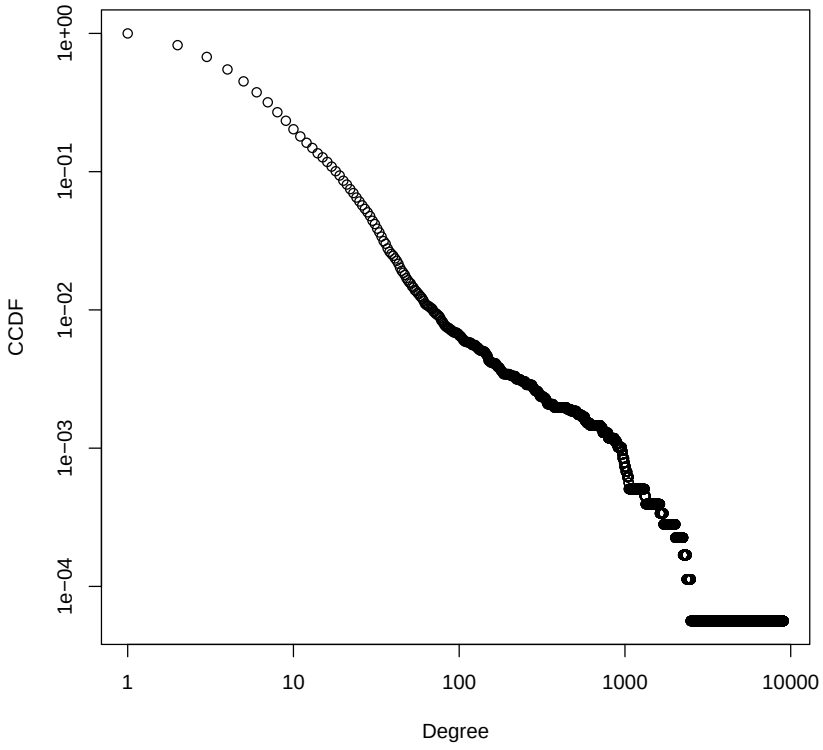


CCDF means Complement Cumulative Distribution Function a.k.a. tail distribution. Reason about these result for a while. What kind of network is this?

11 Compare with other graphs

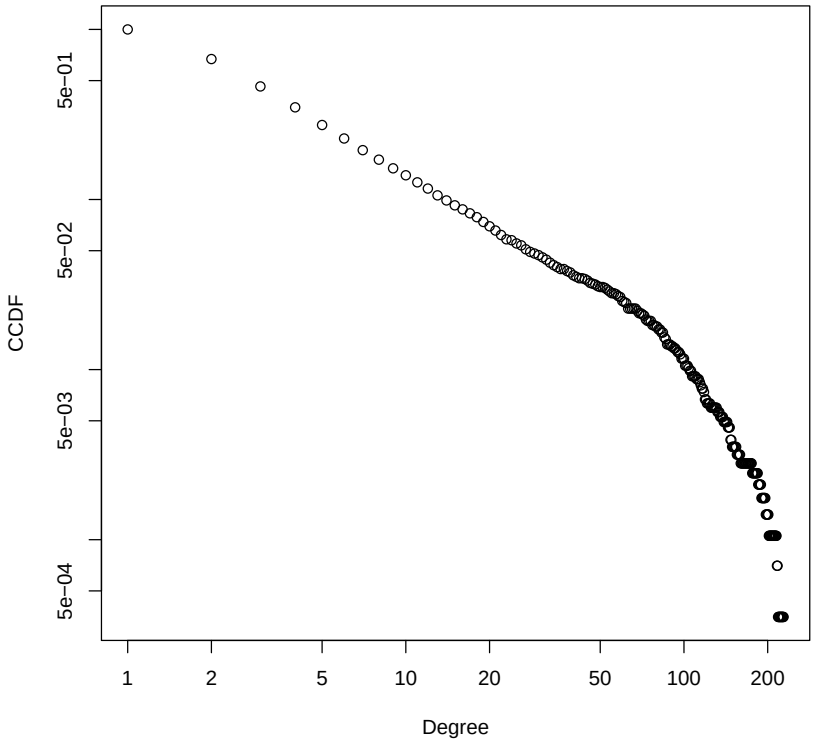
Read debian package dependencies

```
debian<-read.graph("etch_n.txt", "edgelist")
dd_debian<-degree.distribution(debian,mode="total",cumulative=TRUE)
max_degree_debian<-max(degree(debian,mode="total"))
plot(0:max_degree_debian,dd_debian,xlab="Degree",ylab="CCDF",log="xy")
```



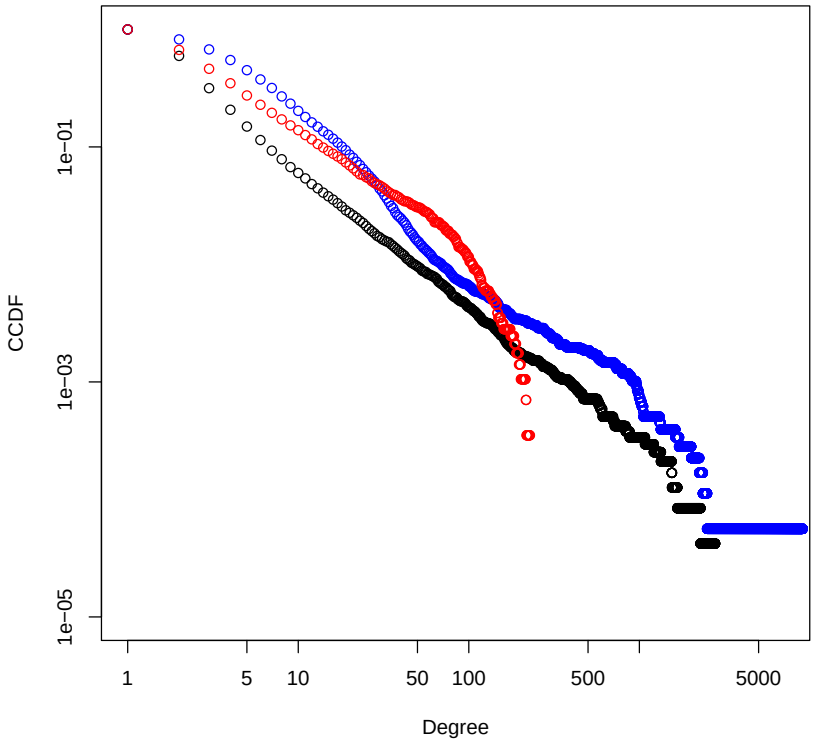
Read airport network

```
airport <-read.graph("airport.gml", format="gml")
dd_air<-degree.distribution(airport,mode="total",cumulative=TRUE)
max_degree_air<-max(degree(airport,mode="total"))
plot(0:max_degree_air,dd_air,xlab="Degree",ylab="CCDF",log="xy")
```



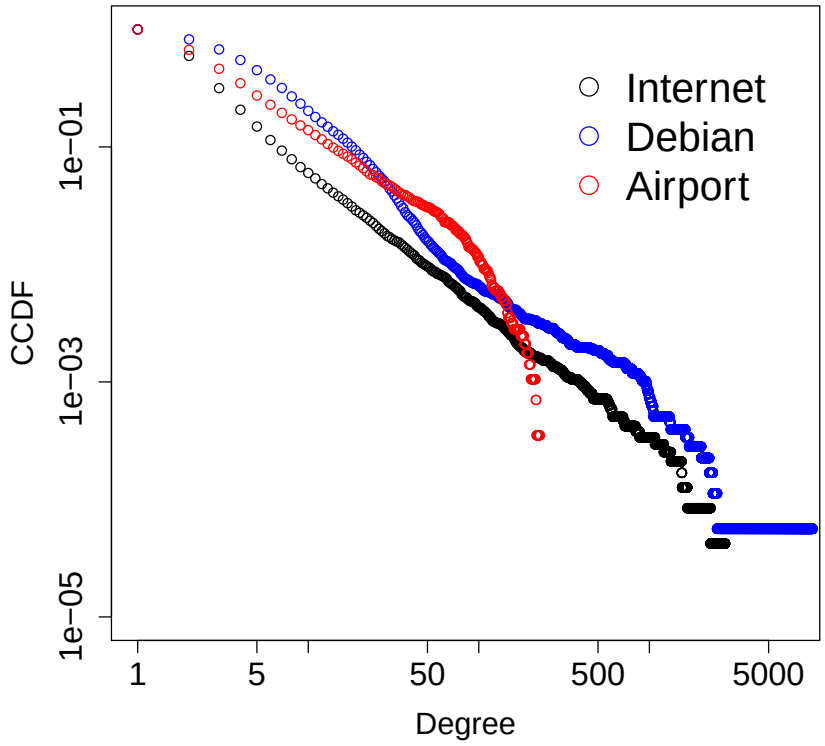
11.1 Plot together

```
plot(1, ylim=c(0.00001,1.0), xlim=c(1,7000), xlab="Degree",
ylab="CCDF",log="xy")
points(0:max_degree_as,dd_as)
points(0:max_degree_debian,dd_debian,col="blue")
points(0:max_degree_air,dd_air,col="red")
```



11.2 Make the plot a bit more fancy

```
plot(1, ylim=c(0.00001,1.0), xlim=c(1,7000), xlab="Degree",ylab="CCDF",
log="xy",cex.axis=1.6, cex.lab=1.5)
points(0:max_degree_as,dd_as)
points(0:max_degree_debian,dd_debian,col="blue")
points(0:max_degree_air,dd_air,col="red")
legend("topright", inset=0.05, legend=c("Internet", "Debian", "Airport"),
pch=c(1,1,1),col=c("black","blue","red"), bty="n", cex=2.0,pt.cex=2)
```



11.3 Compare density

```
ecount(asgraph)
vcount(asgraph)
(ecount(asgraph)*2)/vcount(asgraph)
ecount(airport)
vcount(airport)
(ecount(airport)*2)/vcount(airport)
ecount(debian)
vcount(debian)
(ecount(debian)*2)/vcount(debian)
```

```
[1] 58414
[1] 23748
[1] 4.919488
```

```
[1] 10409
[1] 2845
[1] 7.317399
[1] 92669
[1] 17771
[1] 10.42924
```

11.4 Compare diameter

```
diameter(asgraph,weights=NA)
diameter(airport)
diameter(debian)
```

```
[1] 9
[1] 10
[1] 14
```

11.5 Compare clustering

```
clustering<-transitivity(asgraph,type="local")
asgraph.clustering <- mean(clustering, na.rm=TRUE)
asgraph.clustering
clustering<-transitivity(airport,type="local")
airport.clustering <- mean(clustering, na.rm=TRUE)
airport.clustering
clustering<-transitivity(debian,type="local")
debian.clustering <- mean(clustering, na.rm=TRUE)
debian.clustering
clustering<-transitivity(random.growing.tree,type="local")
tree.clustering <- mean(clustering, na.rm=TRUE)
tree.clustering
N <- 5000
k <- 4
p <- k/N
er <- erdos.renyi.game(N, p)
clustering<-transitivity(er,type="local")
er.clustering <- mean(clustering, na.rm=TRUE)
er.clustering
```

```
[1] 0.6055093
[1] 0.598344
```

```
[1] 0.3842395
[1] 0
[1] 0.0008082019
```

EOF